Final Project Management Science

Arnes Respati Putri
16/392817/PA/17084

Muhammad Ardi Putra
17/415904/PA/18173

November 2019

1 Background

An e-commerce company allows their customers to return damaged products for a new one. In order to do so, the customer has to ship or send the goods to any of the company's hubs or warehouses.

Given a set of customers, hubs, and warehouses data, having values of latitude and longitude of their location. The main objective is to find the shortest route for a customer when he/she is trying to return a product. The end point is a warehouse, but a customer may go directly to the warehouse or to the hub, depending on which is the closest.

There are 30 customers C, 6 hubs H, and 4 warehouses W in the given dataset. In finding the shortest route, a euclidean distance calculation of the latitude and langitude coordinates is used. For one customer, we compare the distance to all hubs and all warehouses and sort the distances ascendingly. Mathematically the euclidean distance calculation is formulated as follows:

$$distance(x, y) = \sqrt{(x_{lat} - y_{lat})^2 + (x_{long} - y_{long})^2}$$

Then we compare the shortest to hub and the shortest to warehouse. If going to a warehouse is closest, then the route is final. Otherwise, we choose the closest hub as the next node in the route. The same calculation is done again, possibly going to other hubs before we get that the next shortest node is the warehouse.

2 Implementation

In computing the shortest route, the Python progamming language is used in the implementation. There are two Python libraries used, namely *pandas* and *math*. The dataset is of the format *.csv* (comma separated values), so the *pandas*

library is used to import and read the .csv file into the program, as can be seen in this line 1 to 2. Then the values are stored in the variables written in line 4 to 21. Furthermore, the *math* library is used to do the euclidean distance calculation where we need to compute the square root of the distance, squared.

```
1  import pandas as pd
2  import math
3
4  df_customers = pd.read_csv("customer_data.csv")
5  df_hubs = pd.read_csv("hub_data.csv")
6  df_warehouses = pd.read_csv("warehouse_data.csv")
7
8  ### Customers data
9  customers_id = list(df_customers["id"].values)
10 customers_latitude = list(df_customers["Latitude"].values)
11 customers_longitude = list(df_customers["Longitude"].values)
12
13 ### Hubs data
14 hubs_name = list(df_hubs["Hub"].values + "_H")
15 hubs_latitude = list(df_hubs["Latitude"].values)
16 hubs_longitude = list(df_hubs["Longitude"].values)
17
18 ### Warehouses data
19 warehouses_name = list(df_warehouses["Warehouse"].values + "_W")
20 warehouses_latitude = list(df_warehouses["Latitude"].values)
21 warehouses_longitude = list(df_warehouses["Longitude"].values)
```

Three different classes are constructed, namely *Customer*, *Hub*, and *Warehouse*. All of the classes have attributes of *name*, *latitude*, and *longitude*. In the Hub class additional attributes of *dist_to_warehouses*, *dist_to_hubs*, *nearest_hub*, *nearest_warehouse*, and *nearest_place* are also defined. *nearest_place* is the one that is chosen as the next node in the route.

The Customer class has the same attributes as in the Hub class, with addition of *route* which includes the nodes (Hubs or Warehouses) that give the shortest cummulative distance.

```
22 class Warehouse:
23     def __init__(self, name, latitude, longitude):
24         self.name = name
25         self.latitude = latitude
26         self.longitude = longitude
27
28 class Hub:
29     def __init__(self, name, latitude, longitude):
30         self.name = name
31         self.latitude = latitude
32         self.longitude = longitude
33         self.dist_to_warehouses = self.dist_to_warehouses() #
       Warehouse name, distance
34         self.nearest_warehouse = self.nearest_warehouse()
35         self.dist_to_hubs = self.dist_to_hubs()
36         self.nearest_hub = self.nearest_hub()
37         self.nearest_place = self.nearest_place()
```

```
38
39      def dist_to_warehouses(self):
40          distances = [[warehouses_name[i], math.sqrt(pow(self.
        latitude - warehouses_latitude[i],2) + pow(self.longitude -
        warehouses_longitude[i],2))] for i in range(len(warehouses_name
        ))]
41          distances.sort(key=lambda x: x[1])
42          return distances
43
44      def nearest_warehouse(self):
45          return self.dist_to_warehouses[0]
46
47      def dist_to_hubs(self):
48          distances = [[hubs_name[i], math.sqrt(pow(self.latitude -
        hubs_latitude[i],2) + pow(self.longitude - hubs_longitude[i],2)
        )] for i in range(len(hubs_name))]
49          distances.sort(key=lambda x: x[1])
50          return distances
51
52      def nearest_hub(self):
53          return self.dist_to_hubs[0]
54
55      def nearest_place(self):
56          if self.dist_to_hubs[1][1] < self.dist_to_warehouses[0][1]:
                # Why not [0][1]? Because we don't wanna compare a hub
         with itself.
57              return self.dist_to_hubs[1]
58          elif self.dist_to_hubs[1][1] > self.dist_to_warehouses
        [0][1]:
59              return self.dist_to_warehouses[0]
60
61  class Customer:
62      def __init__(self, name, latitude, longitude):
63          self.name = name
64          self.latitude = latitude
65          self.longitude = longitude
66          self.dist_to_warehouses = self.dist_to_warehouses()      #
        Warehouse name, distance
67          self.nearest_warehouse = self.nearest_warehouse()
68          self.dist_to_hubs = self.dist_to_hubs()
69          self.nearest_hub = self.nearest_hub()
70          self.nearest_place = self.nearest_place()
71          self.route = self.route()
72
73      def dist_to_hubs(self):
74          distances = [[hubs_name[i], math.sqrt(pow(self.latitude -
        hubs_latitude[i],2) + pow(self.longitude - hubs_longitude[i],2)
        )] for i in range(len(hubs))]
75          distances.sort(key=lambda x: x[1])
76          return distances
77
78      def nearest_hub(self):
79          return self.dist_to_hubs[0]
80
81      def dist_to_warehouses(self):
82          distances = [[warehouses_name[i], math.sqrt(pow(self.
        latitude - warehouses_latitude[i],2) + pow(self.longitude -
```

```
    warehouses_longitude[i],2))] for i in range(len(warehouses_name
    ))]
83          distances.sort(key=lambda x: x[1])
84          return distances
85
86      def nearest_warehouse(self):
87          return self.dist_to_warehouses[0]
88
89      def nearest_place(self):
90          if self.dist_to_hubs[0][1] < self.dist_to_warehouses[0][1]:
91              return self.dist_to_hubs[0]
92          elif self.dist_to_hubs[0][1] > self.dist_to_warehouses
    [0][1]:
93              return self.dist_to_warehouses[0]
94
95      def route(self):
96          loop = True
97          nodes = [[self.name, 0]]
98          nodes.append(self.nearest_place)
99
100         while loop == True:
101             if nodes[-1][0][-1] == "W":
102                 loop = False
103             elif nodes[-1][0][-1] == "H":
104                 for i in range(len(hubs_name)):
105                     if nodes[-1][0] == hubs[i].name:
106                         nodes.append(hubs[i].nearest_place)
107         return nodes
108
109     warehouses = [Warehouse(warehouses_name[i], warehouses_latitude
    [i], warehouses_longitude[i]) for i in range(len(
    warehouses_name))]
110     hubs = [Hub(hubs_name[i], hubs_latitude[i], hubs_longitude[i])
    for i in range(len(hubs_name))]
111     customers = [Customer(customers_id[i], customers_latitude[i],
    customers_longitude[i]) for i in range(len(customers_id))]
112
113 def getAllRoutes():
114         total_travel_distance = 0
115
116         for i in range(len(customers_id)):
117             cumulative_distance = 0
118
119                 print("Customer no", i+1)
120
121                 for j in range(len(customers[i].route)):
122
123                     if customers[i].route[j][0] != customers[i].
    route[-1][0]:
124                         print(customers[i].route[j][0], end=" --> "
    )
125
126                     elif customers[i].route[j][0] == customers[i].
    route[-1][0]:
127                         print(customers[i].route[j][0])
128
129                     cumulative_distance += customers[i].route[j][1]
```

```
130
131                 total_travel_distance += cumulative_distance
132                 print("Cumulative distance =",cumulative_distance,
    "\n\n\n")
133
134     print("Total travel distance =", total_travel_distance)
135
136
137 getAllRoutes()
```

3 Result

The result of the code can be seen in the table 1 below. The routes assigned for all of the customers do not comprised of more than two hubs, which makes the travel cost efficient, in comparison to having to go to 3 or more hubs before arriving to a warehouse.

No.	Customer ID	Routes	Cumulative Dist.
1	e711ce48-95d8-4f38-9fca-a186d18ba497	H_{Cawang} - W_{Cawang}	0.07212381077751406
2	05ef3c47-ee8c-4cff-8dc9-bdeac09f0e52	H_{Bekasi} - H_{Cakung} - W_{Cakung}	0.2201274610878969
3	ed028d3ca-810b-4abf-9997-393dc20f742d	$H_{BogorCitereup}$ - H_{Cawang} - W_{Cawang}	1.1370391613446467
4	ecdbac87-aee1-4bd5-8e21-a6f5d14a76d	H_{Angke} - H_{Cawang} - W_{Cawang}	0.18284084023293654
5	82abb495-0391-4c95-834d-940fa292bb3c	H_{Angke} - H_{Cawang} - W_{Cawang}	0.18186464719753703
6	86fd004a-e6b0-4ef1-9d0d-65881d210d38	H_{Angke} - H_{Cawang} - W_{Cawang}	0.1947730450464499
7	3b65dfaa-42cb-45c0-9b5e-73f2ab2f59e9	H_{Angke} - H_{Cawang} - W_{Cawang}	0.211056893702967
8	24d4e5ab-e313-42bb-bc6b-f1e43d92454a	W_{Cawang}	0.05142353442733911
9	098ed402-eaaa-401b-8779-9bc9d464832b	W_{Cawang}	0.01189860630493703
10	37e14dfc-4473-4b96-b85e-86f929eebac2	H_{Angke} - H_{Cawang} - W_{Cawang}	0.15726802238056328
11	331828ed-3bb3-42f9-b0af-f0a639a1ffc6	H_{Cakung} - W_{Cakung}	0.059125400016359714
12	161a88ae-9021-4ce9-8a1c-3fcc38752a11	H_{Cawang} - W_{Cawang}	0.08957756387854611
13	cf0c521c-db65-4b54-8f12-64a3d33d2113	H_{Cawang} - W_{Cawang}	0.07451445429966237
14	fd03a70b-2b2f-4e8e-88c3-cf6d6d4a60a3	H_{Angke} - H_{Cawang} - W_{Cawang}	0.17985801972171264
15	fefd46d6-d2d0-45e8-a054-131fbeda8e74	H_{Cawang} - W_{Cawang}	0.04997274771870781
16	75410b2d-1918-4a0f-b2b7-720ce8c4e92a	H_{Cawang} - W_{Cawang}	0.05073349252256351
17	deeb2897-3a4f-4471-80a3-719bd73580d9	H_{Cawang} - W_{Cawang}	0.1168712989696367
18	cc745d50-a0b3-4995-adc4-42e09f812250	W_{Ceper}	0.07669772716320501
19	de908962-149c-457a-b27b-5ce9a4e808f9	H_{Angke} - H_{Cawang} - W_{Cawang}	0.19840942066228826
20	d4a2be90-2c5c-490d-baab-18a7d9e2e896	H_{Angke} - H_{Cawang} - W_{Cawang}	0.22257780401597188
21	66a7c70b-d026-4583-a119-745afc43e60a	H_{Angke} - H_{Cawang} - W_{Cawang}	0.21594209111745197
22	e6c4e01c-2331-42de-81ba-36b80ff3bb5f	H_{Cakung} - W_{Cakung}	0.0780288964465363
23	a4d052db-22cd-4754-ac1b-6b13eb36c5cf	H_{Angke} - H_{Cawang} - W_{Cawang}	0.17634209626110403
24	e540732d-4292-4db1-bc6d-208a37ad20ab	$H_{BogorCitareup}$ - H_{Cawang} - W_{Cawang}	1.255114506507194
25	1cebf438-11c4-4137-982a-7862da1bae90	H_{Cawang} - W_{Cawang}	0.13135099715152718
26	8f54d942-0595-41ce-a94a-74f73bc43495	H_{Cawang} - W_{Cawang}	0.09447383312412896
27	8c4e348c-f586-428c-bcc5-91140022b8dd	H_{Cawang} - W_{Cawang}	0.13276624296696432
28	9f4bc659-bba0-425a-bf46-7abedef98a99	$H_{Karawaci}$ - $W_{Karawaci}$	0.5157044018370592
29	c7cf71e1-a673-4a1a-8ace-cfecc8b70a90	W_{Ceper}	0.06395311784425624
30	377e1d8d-5893-4e15-a073-dd8428b34862	W_{Ceper}	0.06809304132582611
Total travel distance			6.27052317605349

Table 1: Routes for each customer